Operating Systems (2INC0)

Memory Management (09)
Reminders

Dr. Geoffrey Nelissen
Courtesy of Dr. Tanir Ozcelebi,
Dr. I. Radovanovic, and Dr. R. Mak
(figures from Bic & Shaw)

IRiS
Interconnected
Resource-aware
Intelligent Systems

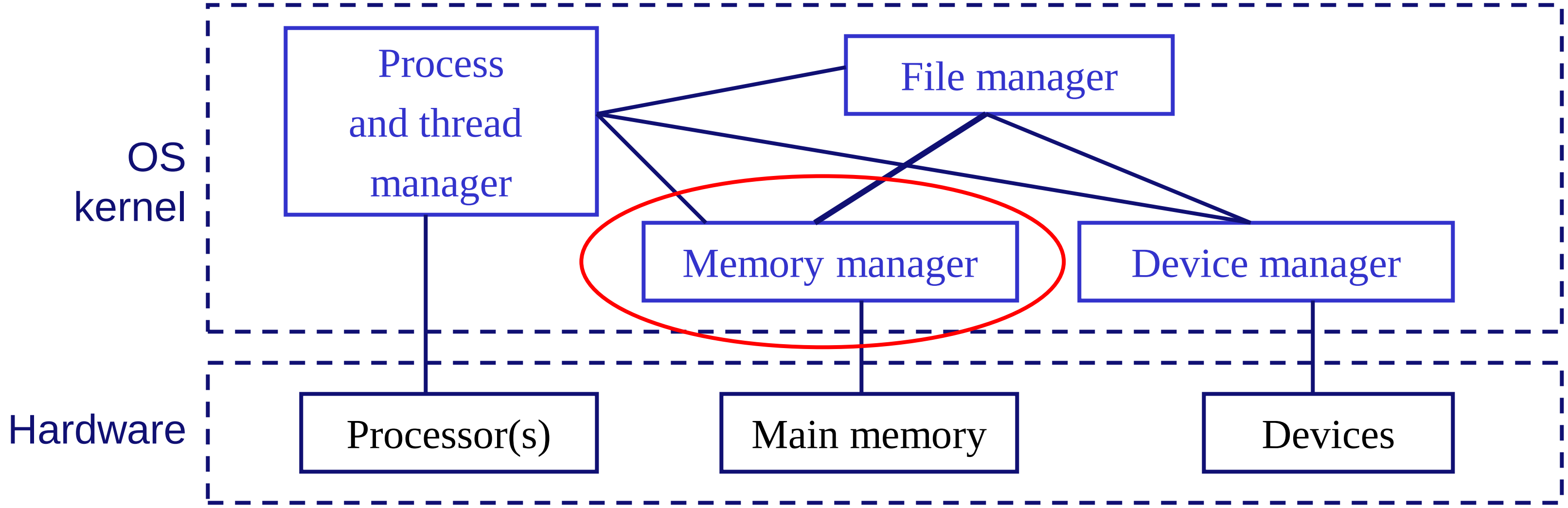
TU/e Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

OS as resource manager

- Resource: Anything that is needed for a process to run
  - Main Memory (MM)
  - Space on a disk
  - CPU
- **OS roles:**
  - create resource **abstractions** (hide HW details)
    - files for disk space, processes for memory use, threads for CPU use, …
  - **manage resource sharing**
    - space-sharing vs time-sharing
    - provides a mechanism to isolate (protect) resources and programs.

Logical OS Organization



- All modules interact to coordinate their activities.
- Examples:
  - For virtual memory: Process Man. – Memory Man.
    - Coordinate scheduling activity with memory allocation
  - For performance improvement: File Man. – Memory Man.
    - Information from storage device read prior to request by a thread

Process execution

- Requirements:
  - A running program must be brought (from disk) into MM.
  - Each process must have a distinct address space (protection).

- ➔ Main memory is shared in time and space between concurrent programs

User/programmer perspective

A good memory is one that…

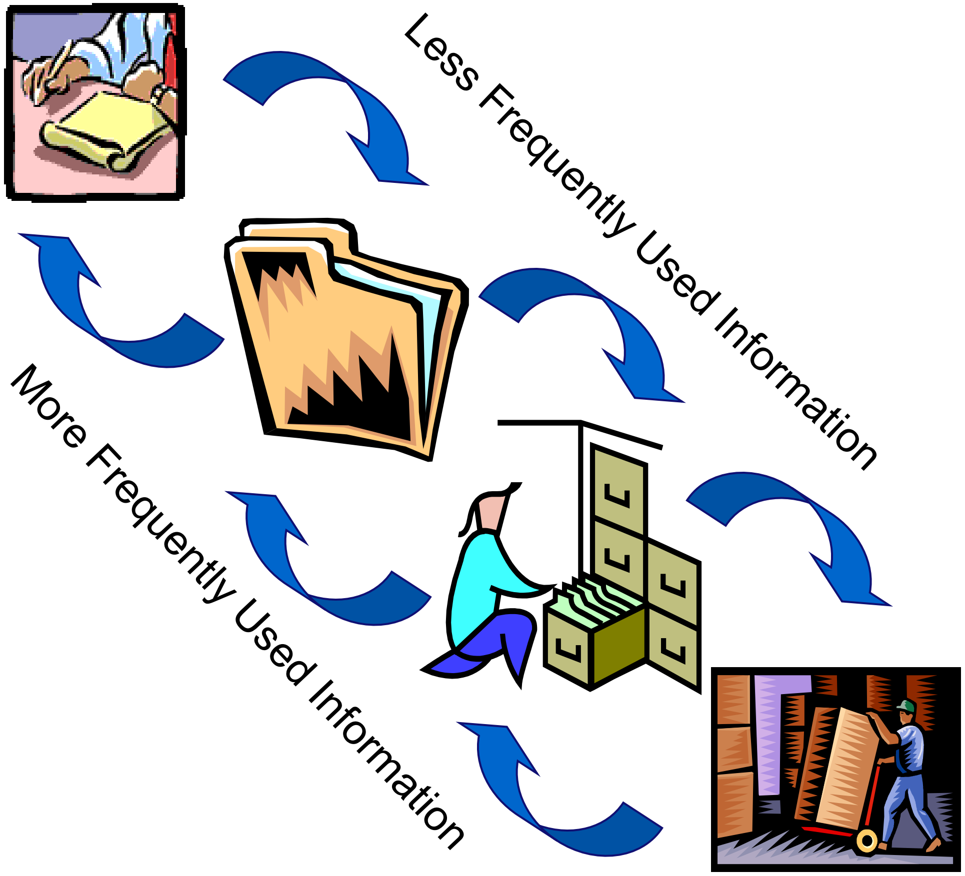
- …is non-volatile
- …is private for a program (protected)
- …has infinite capacity
- …has zero access time
- …is simple to use
- …is cheap

Such a memory clearly does not exist, …

- … but a hierarchical hardware organization combined with virtualization can help!
  - We discuss virtual memory in the next lecture…

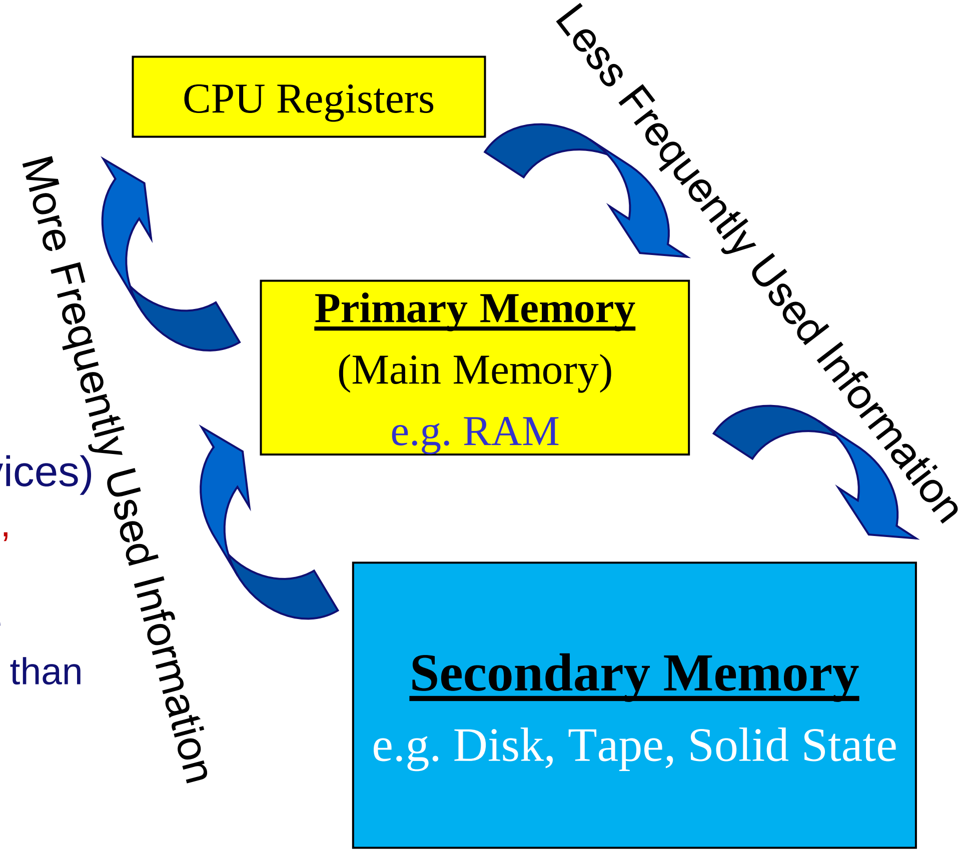
Analogy:
Office storage hierarchy

- Office storage hierarchy
  - Desktop
  - File folder
  - File cabinet
  - Warehouse
- Upper-layers:
  - info more frequently used
- Lower layers:
  - info less frequently used

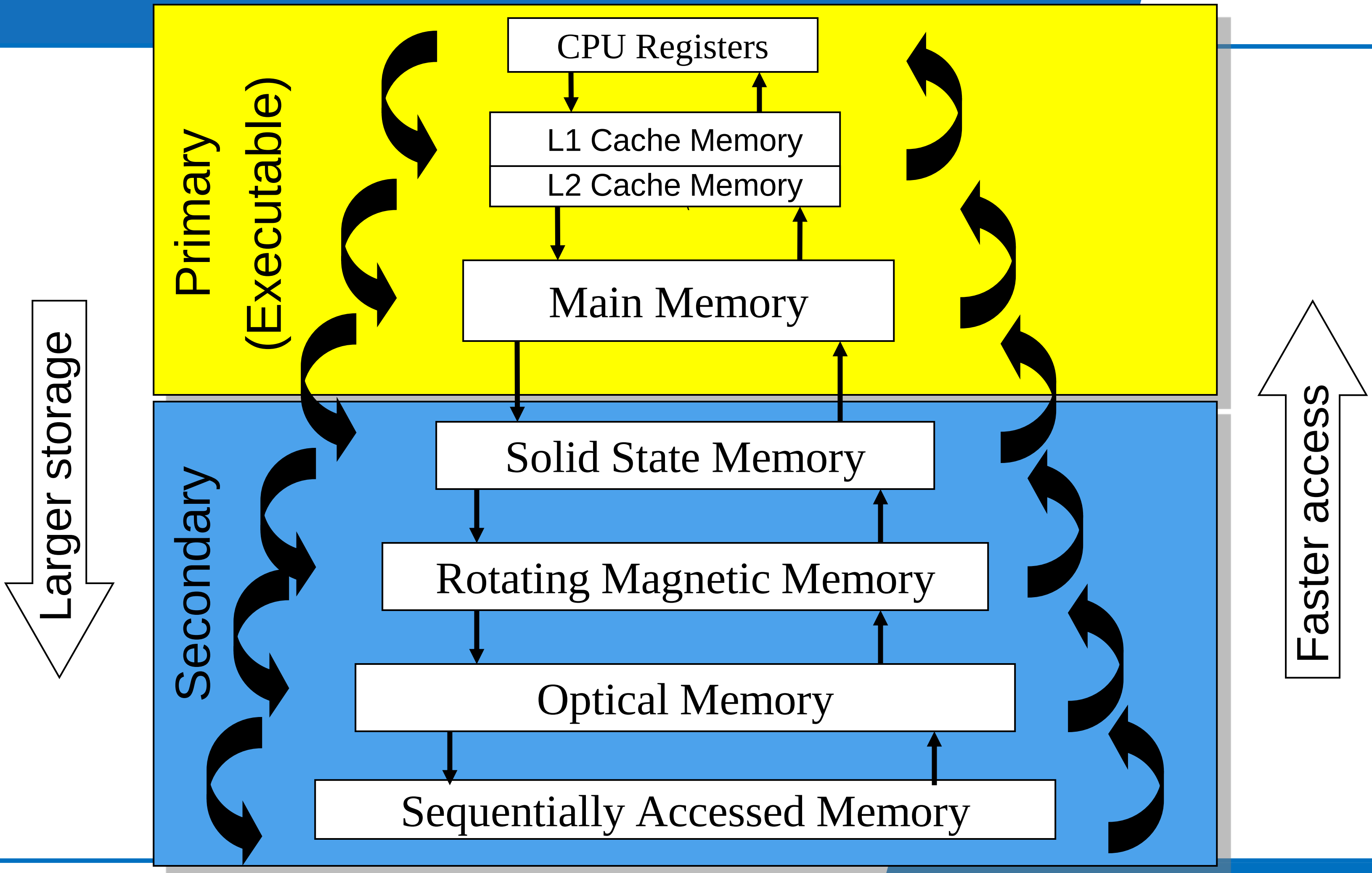


Memory hierarchy

- **CPU registers**
  - **fast access (1 clk cycle)**
  - **limited in size**
- **Primary (executable) memory**
  - **direct access (~a 100 clk cycles),**
  - **relatively larger**
- **Secondary memory** (storage devices)
  - **Accessed through I/O operations**,
  - **very cheap / very large**
  - data stored for a long period of time
  - **slow** (3 orders of magnitude slower than CPU register memory)



Contemporary memory hierarchy & dynamic loading



OS exploiting the hierarchy

- **Place**
  - **frequently-used info high in the hierarchy**
  - **infrequently-used info low in the hierarchy**

OS exploiting the hierarchy

- **Place**
  - **frequently-used info high in the hierarchy**
  - **infrequently-used info low in the hierarchy**

- Updates are first applied to upper memory.
  - Upward moves are (usually) *copy* operations
    - Image exists in both higher & lower memories
  - Downward moves are (usually) *destructive*
    - Usually to save space in upper memory.
    - Destroy image in upper memory
    - Update image in lower memory
      - avoid when the copy in lower memory is still consistent!

Memory manager functional requirements

- Classical memory managers provide
  - **Abstraction**
    - **MM** (virtually) **appears to be larger** than the physical machine mem.
    - Memory is presented as an **array of contiguously addressed bytes**
    - Abstract set of **logical addresses** to reference physical memory
  - **Allocation**
    - Allocate primary memory to processes as requested
  - **Isolation**
    - Enable **mutually exclusive use of memory** by processes
  - **Memory Sharing**
    - **Enable memory to be shared by processes**

Operating Systems (2INC0)

Memory Management (09)
Using partitions

Dr. Geoffrey Nelissen

Courtesy of Dr. Tanir Ozcelebi,
Dr. I. Radovanovic, and Dr. R. Mak
(figures from Bic & Shaw)

IRiS
Interconnected
Resource-aware
Intelligent Systems

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

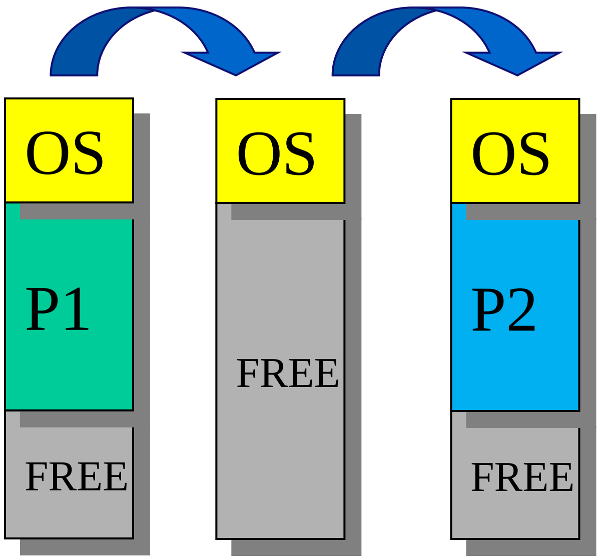
Early systems

- **Single-programmed solution**
- Multi-programmed solutions
  - Fixed partitions
  - Dynamic partitions
  - Relocatable dynamic partitions

Single-programmed solution

- **Program loaded in its entirety into MM** and allocated as much contiguous MM space as needed
- Once in MM, program **stays there until execution ends**.

- **Problems:**
  - **No multiprogramming support.**
  - **Large context-switch overhead**
  - **If a program does not fit in the memory, it cannot be executed.**



- **Advantages:**
  - **simple hardware** requirements (base address and limit registers)
  - **Easy protection** between OS and processes and between processes

Early systems

- Single-programmed solution
- Multi-programmed solution
  - **Fixed partitions**
  - Dynamic partitions

Fixed partitions

- **Divide the memory into fixed partitions**
  - **one process per partition**
  - partitions may have different sizes to accommodate various types of programs (fixed at OS initialization)
  - a process can only access its own partition (protection)

- **Partition allocation scheme** needed (next slide uses first-fit)

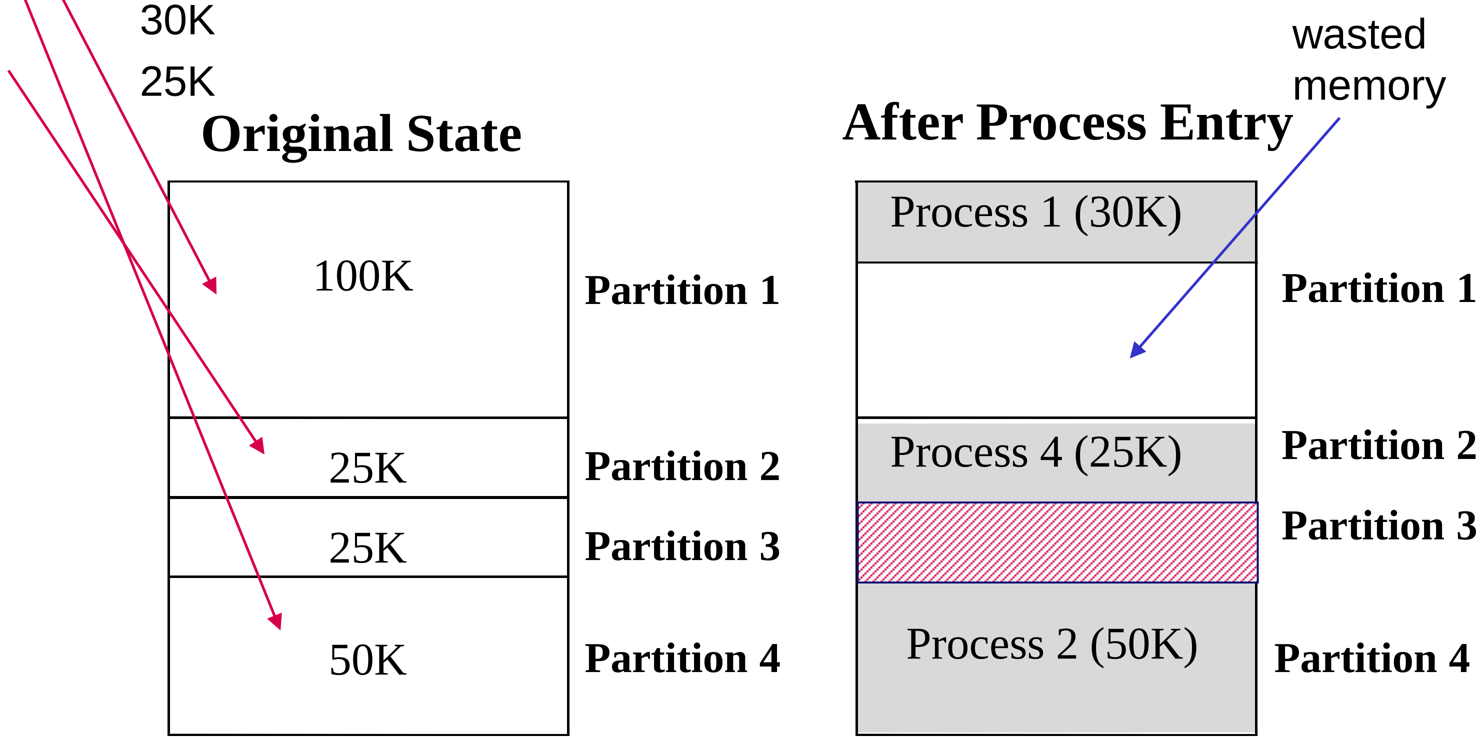
- **Disadvantage**:
  - Entire program must be stored in MM
  - **Internal fragmentation**
    - Solution: Choosing the right partition size
      - Too small – long turnaround time (doesn't fit in any partition in the worst case)
      - Too big – wasted memory

Fixed partition Example

Process List :

P1 30K
P2 50K
P3 30K
P4 25K

Process 3 must wait even though **70K of contiguous free space** is available (**and 70K+25K available in total**).



Early systems

- Single-programmed configuration
- Multi-programmed configurations
  - Fixed partitions
  - **Dynamic partitions**
    - Allocation
    - De-allocation

Dynamic partitions

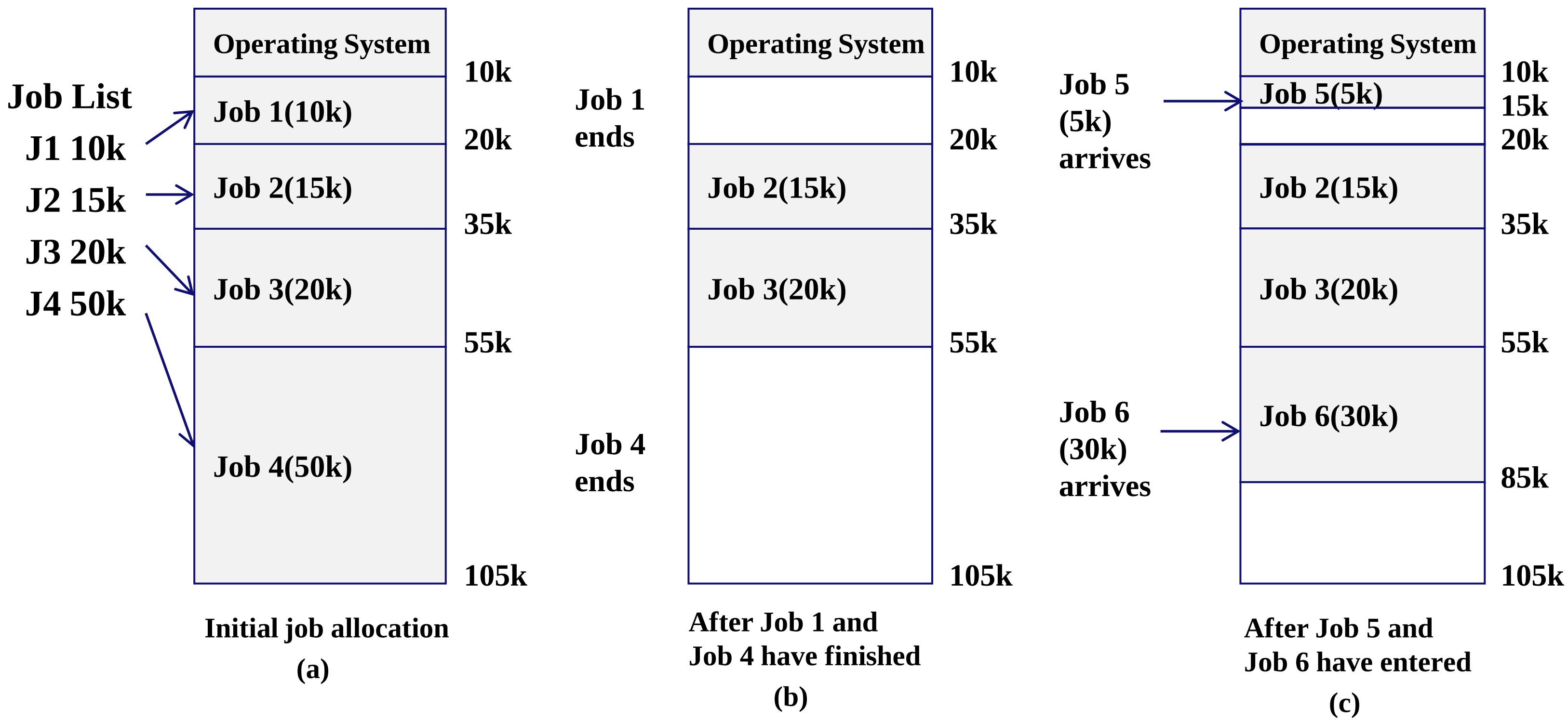
- **Partition size** not fixed but determined **based on the process size.**

- **Advantage:**
  - **no** memory waste due to **internal fragmentation**

- **Disadvantages:**
  - **External fragmentation**
    - Memory utilization deteriorates as new processes enter and old processes exit the system
  - **The complete program must still be stored contiguously.**

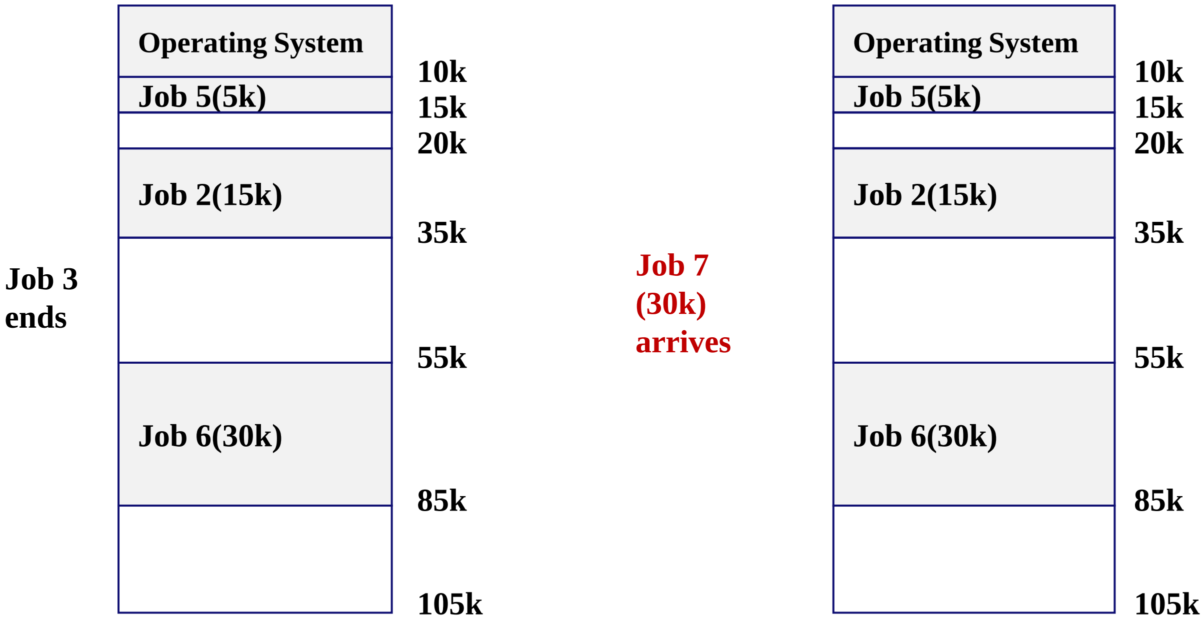
Dynamic partitions Example

First-fit allocation scheme



Dynamic partitions
Example

First-fit allocation scheme (cont'd)



After Job 3 has finished
(d)

After Job 7 has entered
(e)

Job 7 must wait even though there is **45K of free memory** space available

Dynamic partition: Allocation schemes

- Goals: decrease allocation time, increase memory utilization

- First-fit
  - Allocate the *first* partition that is big enough
  - Advantage:
    - Faster in making the allocation
  - Disadvantage:
    - Waste more memory space

- Best-fit
  - Allocate the smallest partition fitting the requirements
  - Advantage:
    - Produces the smallest leftover partition
    - Makes best use of memory *when it is possible to allocate a new job*
  - Disadvantage:
    - Takes more time

Memory allocation
Keep track of holes using a list

Memory location	Memory block size
10240	30K
40960	15K
56320	50K
107520	20K

Total Available: 115K

Memory allocation
First-fit

Order the list by
memory location

Memory location	**Memory block size**
10240	30K
40960	15K
56320	50K
107520	20K

Total Available: 115K

Memory allocation
Best-fit

Order the list by
block sizes

Memory location	**Memory block size**
40960	15K
107520	20K
10240	30K
56230	50K

Total Available: 115K

Exercise

- The table of free memory spaces is given below. Show how this table will look after allocating a process that occupies 200 spaces using the *Best-Fit algorithm*.

Before request		*After request*	
Beginning address	**Memory block size**	**Beginning address**	**Memory block size**
4075	105	4075	105
5225	5	5225	5
6785	600	6785	600
7560	20	7560	20
7600	205	*7800	5
10250	4050	10250	4050
15125	230	15125	230
24500	1000	24500	1000

Best-Fit is **not** the best for dynamic partitions: Excessive memory fragmentation due to smaller free space

Allocation schemes

- First-Fit: clusters small holes at the top
- Next-Fit (rotating First-Fit):
  - start each search at the point where the previous search stopped

    (faster search, slightly worse memory utilization than First-Fit)
- Best-Fit: worst performance due to excessive fragmentation
- Worst-Fit: always uses the largest available partition
  - similar performance to first-fit and next-fit
- Optimized schemes
  - statistics on average process size etc. needed

Early systems

- Single-programmed configuration
- Multi-programmed configurations
  - Fixed partitions
  - **Dynamic partitions**
    - Allocation
    - **De-allocation**

Release of memory space

- Occurs **when process terminates or suspends**

- For **fixed partitions**:
  - Memory Manager **resets status of memory block to “free”.**
- For **dynamic partitions** tries to **combine free areas** of memory
  - Is the block **adjacent** to another free block?
    - If yes, combine the 2 blocks

  - Is the block **between** 2 free blocks?
    - If yes, combine those 3 blocks

  - Is the block **isolated** from other free blocks?
    - If yes, make a new table entry

De-allocation Example

	Before Deallocation		
	Beginning address	**Memory block size**	**Status**
	4075	105	Free
	5225	5	Free
	6785	600	Free
	7560	40	Free
De-allocate →	(7600)	(200)	(Busy)[1]
	*7800	5	Free
	10250	4050	Free
	15125	230	Free
	24500	1000	Free

Important note

- **De-allocate only processes that are <u>really idle</u>**

- **Example: not waiting for an I/O device** to write in an I/O buffer in the process address space
➔ otherwise, the **I/O manager may write in the address space of the wrong process**

Operating Systems (2INC0)

Memory Management (09)
Relocatable partitions

Dr. Geoffrey Nelissen

Courtesy of Dr. Tanir Ozcelebi,
Dr. I. Radovanovic, and Dr. R. Mak
(figures from Bic & Shaw)

Interconnected
Resource-aware
Intelligent Systems

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Relocatable dynamic partitions

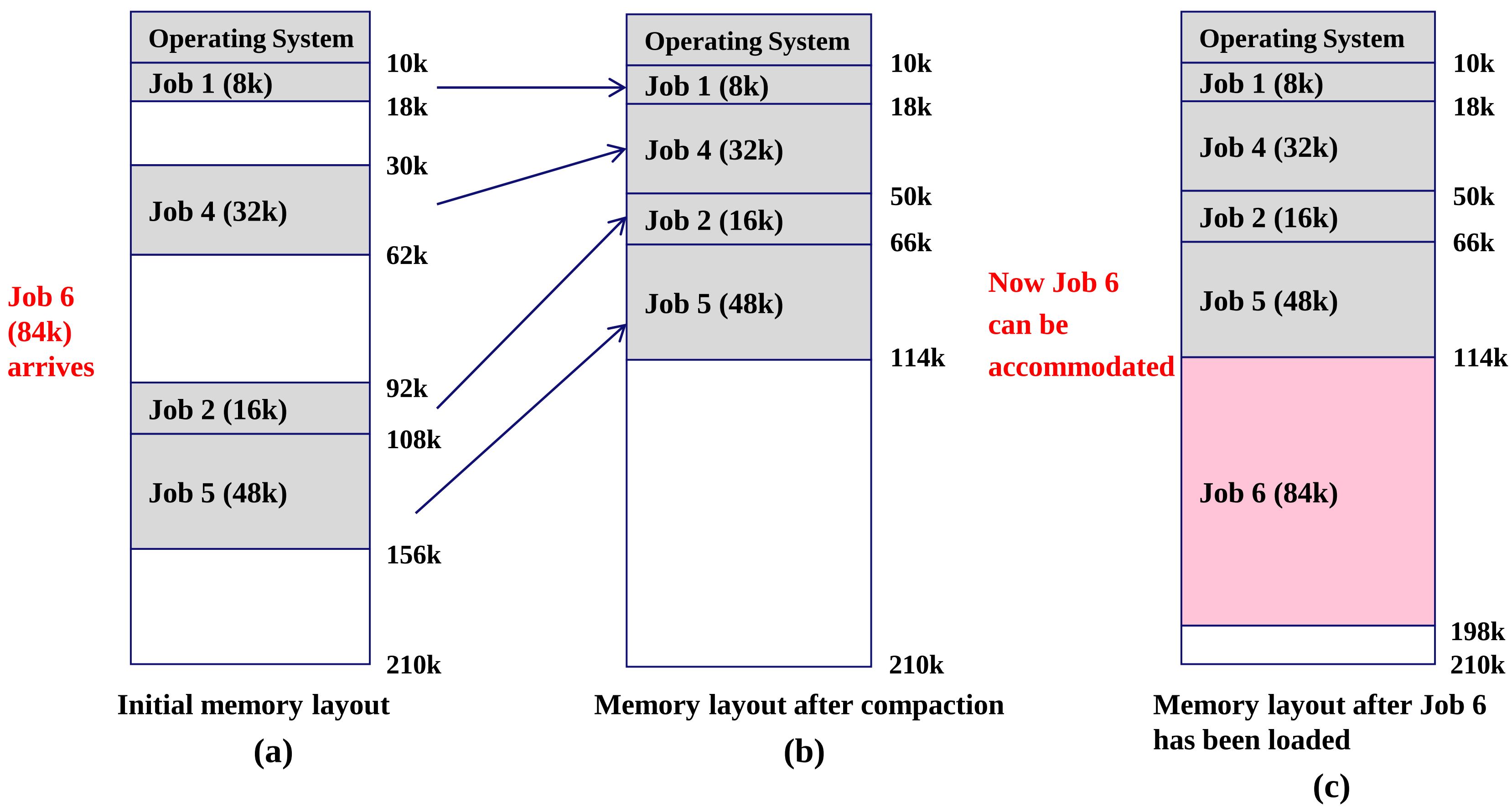
- *Memory manager* **relocates programs to gather all empty blocks** and generate one big free memory block.

- **Solves both internal and external fragmentation problems**
  - relocation and compaction **avoids memory waste**
  - "insufficient memory" problems less frequent

Compaction steps

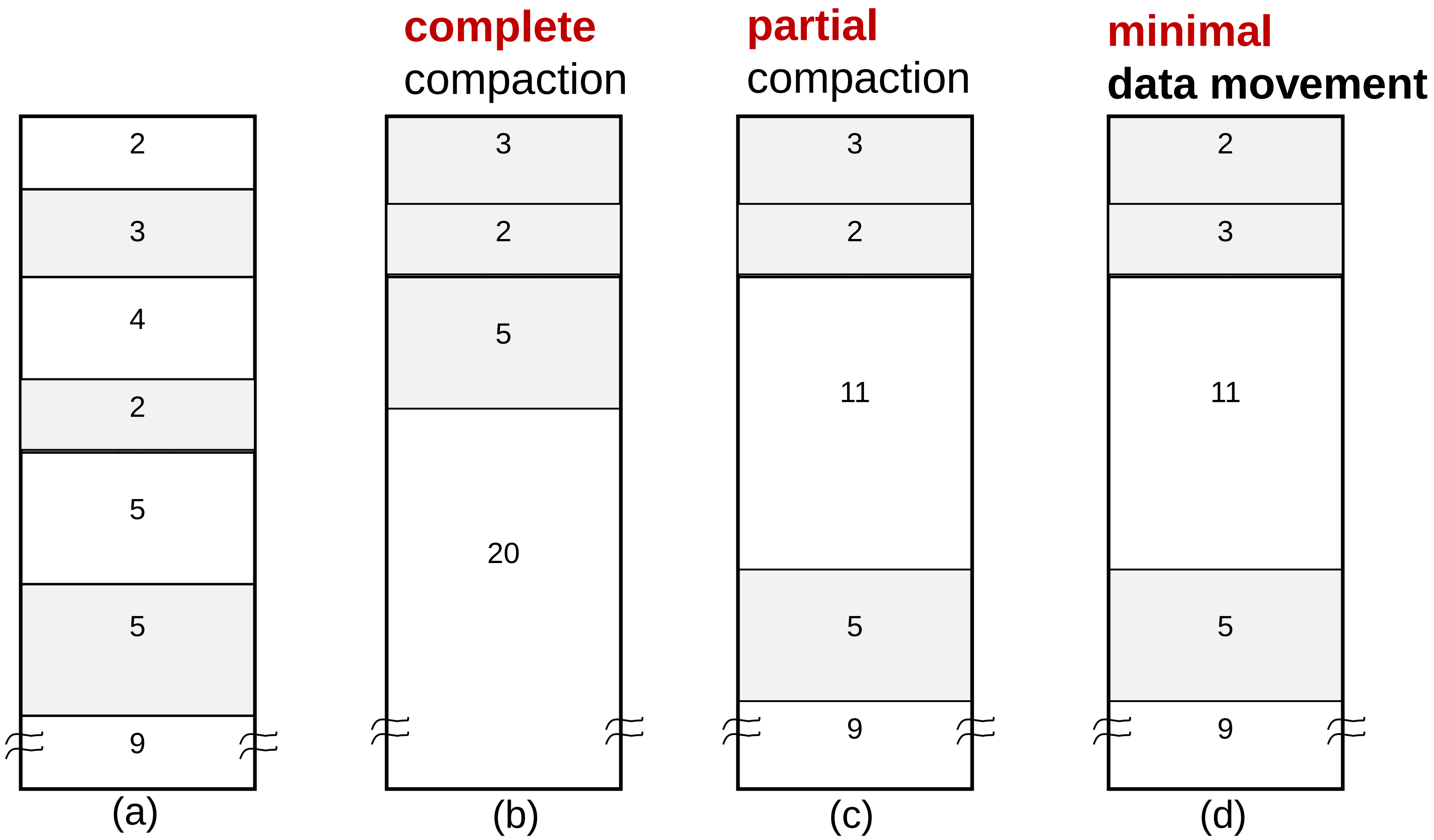
- **Relocate** every program in memory so that they're **contiguous**.
- **Adjust every address**, and every reference to an address, within each program to account for programs' new locations in memory.
  - Other values within the program are kept unchanged (e.g., data values).
- **Question: When should compaction be done?**
  - When a certain percent of memory becomes busy (e.g. 75%),
  - When there are processes pending, or
  - After a prescribed amount of time
- **Note: Compaction cannot be done if address binding is static.**

Compaction Example



Memory compaction (strategies)

We need a block of size 10



Memory compaction (remarks)

- **Disadvantage**: **very costly overhead**
  - requires each word to be read from and written into memory
    - May take several seconds

- Contemporary systems
  - **Virtual Memory techniques** (paging, segmentation) instead of compaction

Relocation

Q: How to keep track of where each process is with respect to its original location?

A:

- Special-purpose registers
  - Relocation register
    - Contains value to be added to each address referenced in the program
  - Limit register
    - Stores the size of the memory space accessible by each program

Swapping

- When?
  - a program must be loaded into MM and there is not enough room
  - move something else to secondary storage to make room.

- How?
  - OS selects a process to swap-out and replaces it with another process from the secondary storage
  - swap-out only those parts of process space that are not already on disk
    - typically code is not modified
    - benefits from hardware support that registers (keeps track of) modification

- Where to (on secondary memory)?
  - either to an arbitrary file (introduces file management overhead)
  - or to a special partition on disk, the *swap space*
    - more efficient (low-level I/O)